# Złożoność obliczeniowa

wykład 12

# Obliczenia alternujące

- Alterternujące maszyny Turinga (AMT) to uogólnienie maszyn niedeterministycznych (NMT)
- Zarówno NMT jak i AMT nie są realistycznym modelem obliczeń. NMT pomagają nam jednak zauważyć bardzo naturalne zjawisko: różnicę między znalezieniem rozwiązania a możliwością jego zweryfikowania.
- Podobną rolę pełnią AMT dla pewnych języków, dla których nie ma krótkich świadków, więc które nie mogą być scharakteryzowane samym niedeterminizmem.

# Obliczenia alternujące

Definicja AMT:

- konfiguracja może mieć wielu następników (podobnie jak w NMT)
- dodatkowo stany maszyny (w efekcie także konfiguracje) są podzielone na egzystencjalne i uniwersalne

# Obliczenia alternujące

Definicja AMT:

- konfiguracja może mieć wielu następników (podobnie jak w NMT)
- dodatkowo stany maszyny (w efekcie także konfiguracje) są podzielone na egzystencjalne i uniwersalne

Zbiór konfiguracji wygrywających to najmniejszy zbiór taki, że:

- konfiguracja akceptująca jest wygrywająca
- konfiguracja egzystencjalna, której pewien następnik jest wygrywający, jest wygrywająca
- konfiguracja uniwersalna, której każdy następnik jest wygrywający, jest wygrywająca

Akceptujemy słowo $w$, jeśli konfiguracja początkowa dla tego słowa jest wygrywająca.

$M$ działa w czasie $T(n)$ / pamięci $S(n)$, jeśli każde obliczenie działa w tym czasie / pamięci.

# Obliczenia alternujące

Definicja AMT:

- konfiguracja może mieć wielu następników (podobnie jak w NMT)
- dodatkowo stany maszyny (w efekcie także konfiguracje) są podzielone na egzystencjalne i uniwersalne

Zbiór konfiguracji wygrywających to najmniejszy zbiór taki, że:

- konfiguracja akceptująca jest wygrywająca
- konfiguracja egzystencjalna, której pewien następnik jest wygrywający, jest wygrywająca
- konfiguracja uniwersalna, której każdy następnik jest wygrywający, jest wygrywająca

Akceptujemy słowo $w$, jeśli konfiguracja początkowa dla tego słowa jest wygrywająca.

$M$ działa w czasie $T(n)$ / pamięci $S(n)$, jeśli każde obliczenie działa w tym czasie / pamięci.

Spostrzeżenie:
NMT to szczególny przypadek AMT – same stany egzystencjalne

# Obliczenia alternujące

Równoważnie: akceptację można zdefiniować za pomocą gry:

- patrzymy na graf konfiguracji (krawędzie = możliwe przejścia)
- gracze ∃ i ∀ na przemian przesuwają (wspólny pionek) po tym grafie
- w stanie egzystencjalnym decyduje gracz ∃, w uniwersalnym ∀ (gracz ∃ stara się zaakceptować, gracz ∀ odrzucić)
- słowo akceptujemy, jeśli gracz ∃ ma strategię wygrywającą – może dojść do konfiguracji akceptującej

# Obliczenia alternujące

Klasy **ATIME**($T(n)$), **ASPACE**($S(n)$),

**AP**=$\cup_k$ **ATIME**($n^k$), **AL**=**ASPACE**($log\ n$)

Twierdzenie

**AL**=**P**, **AP**=**PSPACE** (można też bardziej ogólnie)

# Obliczenia alternujące

Klasy **ATIME**($T(n)$), **ASPACE**($S(n)$),

**AP**=$\cup_k$ **ATIME**($n^k$), **AL**=**ASPACE**($log\ n$)

Twierdzenie

**AL**=**P**, **AP**=**PSPACE** (można też bardziej ogólnie)

Dowód **AP**$\subseteq$**PSPACE**

Backtracking: przeglądamy wszystkie możliwe biegi maszyny alternującej (zapis takiego biegu mieści się w pamięci wielomianowej)

# Obliczenia alternujące

Klasy **ATIME**($T(n)$), **ASPACE**($S(n)$),

**AP**=$\cup_k$ **ATIME**($n^k$), **AL**=**ASPACE**($log\ n$)

Twierdzenie

**AL**=**P**, **AP**=**PSPACE** (można też bardziej ogólnie)

Dowód **AL**$\subseteq$**P**

Tworzymy graf wszystkich konfiguracji maszyny alternującej – jest on rozmiaru wielomianowego. W czasie wielomianowym można zaznaczyć wszystkie konfiguracje wygrywające, idąc od tyłu (od konfiguracji akceptujących).

# Obliczenia alternujące

Klasy **ATIME**($T(n)$), **ASPACE**($S(n)$),
**AP**=$\cup_k$ **ATIME**($n^k$), **AL**=**ASPACE**($log\ n$)

Twierdzenie

**AL**=**P**, **AP**=**PSPACE** (można też bardziej ogólnie)

Dowód **PSPACE**$\subseteq$**AP**

Wystarczy pokazać, że QBF$\in$**AP**, bo QBF jest **PSPACE**-zupełny.
A to jest w zasadzie oczywiste – gracz $\exists$ decyduje o wartości zmiennych kwantyfikowanych egzystencjalnie, gracz $\forall$ o wartości zmiennych kwantyfikowanych uniwersalnie, na koniec deterministycznie liczymy wartość formuły.
Tak naprawdę: algorytm dla **AP** jest prostszy niż dla **PSPACE**.

# Obliczenia alternujące

Klasy **ATIME**($T(n)$), **ASPACE**($S(n)$),

**AP**=$\cup_k$ **ATIME**($n^k$), **AL**=**ASPACE**($log\ n$)

Twierdzenie

**AL**=**P**, **AP**=**PSPACE** (można też bardziej ogólnie)

Dowód **P**$\subseteq$**AL**

- Dla algorytmu w **P** istnieje równoważny obwód logiczny, który umiemy skonstruować w pamięci logarytmicznej.
- Łatwo podać algorytm w **AL**, który wyznacza wartość obwodu: gracze idą od bramki wynikowej, w bramce OR gracz $\exists$ decyduje, które wejście jest prawdziwe, w bramce AND gracz $\forall$ decyduje, które wejście miałoby być fałszywe.
- Nie generujemy całego obwodu, tylko poszczególne fragmenty, „na żądanie”.

# Obliczenia alternujące

Klasy **ATIME**($T(n)$), **ASPACE**($S(n)$),

**AP**=$\cup_k$ **ATIME**($n^k$), **AL**=**ASPACE**($log\ n$)

Twierdzenie

**AL**=**P**, **AP**=**PSPACE** (można też bardziej ogólnie)

Porównanie między **AP** a **IP** (dowodami interaktywnymi):

- gracz $\exists$ $\leftrightarrow$ prover, gracz $\forall$ $\leftrightarrow$ verifier
- W **AP** gracz $\forall$ dysponuje niedeterminizmem, w **IP** weryfikator to zwykły algorytm (deterministyczny, z dostępem do losowości)

# Złożoność a kryptografia

- Podstawowy cel kryptografii: zaszyfrować wiadomość tak, aby przeciwnik nie umiał jej odszyfrować
- O przeciwniku powinniśmy myśleć jako o urządzeniu o ograniczonej mocy obliczeniowej – np. RSA: zaszyfrowana wiadomość i klucz publiczny jednoznacznie wyznaczają oryginalną wiadomość, ale nie umiemy jej szybko wyliczyć
- Zatem: chcemy, aby odszyfrowanie wiadomości sprowadzało się do (tzn. było nie łatwiejsze niż) rozwiązania jakiegoś problemu trudnego obliczeniowo

# Złożoność a kryptografia

- Dowód bezpieczeństwa musi opierać się co najmniej na hipotezie że **P≠NP** (lub na jakiejś mocniejszej) – dysponując niedeterminizmem łatwo można odszyfrować, zgadując oryginalną wiadomość, więc jeśli **P=NP**, to możemy szybko odszyfrowywać deterministycznie

# Złożoność a kryptografia

- Dowód bezpieczeństwa musi opierać się co najmniej na hipotezie że **P≠NP** (lub na jakiejś mocniejszej) – dysponując niedeterminizmem łatwo można odszyfrować, zgadując oryginalną wiadomość, więc jeśli **P=NP**, to możemy szybko odszyfrowywać deterministycznie
- Początkowo próbowano oprzeć bezpieczeństwo szyfrowania na trudności jakiegoś problemu **NP**-zupełnego (czyli na hipotezie **P≠NP**). To się do tej pory nie udało.
- Powód (w skrócie): **NP**-zupełność mówi o trudności problemów dla najgorszych danych wejściowych. Natomiast w kryptografii potrzebujemy problemów, które są trudne dla większości danych wejściowych. Jeśli coś szyfrujemy, to przeciwnikowi powinno być trudno odszyforwać (prawie) każdą zaszyfrowaną wiadomość, a nie tylko niektóre (najtrudniejsze).

# Złożoność a kryptografia

Trudne problemy, na których bazuje bezpieczeństwo protokołów kryptograficznych:

- Rozkładanie liczb złożonych. Najlepszy znany algorytm działa w czasie około $2^{O(n^{1/3})}$. Trudno jest rozkładać liczby postaci $p \cdot q$, gdzie $p$ i $q$ to liczby pierwsze podobnych rozmiarów.
- Logarytm dyskretny. Niech $g$ będzie generatorem grupy $\mathbb{Z}_p^*$. Znając $(g^x \bmod p)$ trudno jest wyznaczyć $x$. Można pokazać, że jeśli ten problem jest trudny w najgorszym przypadku, to jest też trudny w średnim przypadku.
- I inne (np. bazujące na krzywych eliptycznych)

# Złożoność a kryptografia

Trudne problemy, na których bazuje bezpieczeństwo protokołów kryptograficznych:

- Rozkładanie liczb złożonych. Najlepszy znany algorytm działa w czasie około $2^{O(n^{1/3})}$. Trudno jest rozkładać liczby postaci $p \cdot q$, gdzie $p$ i $q$ to liczby pierwsze podobnych rozmiarów.
- Logarytm dyskretny. Niech $g$ będzie generatorem grupy $\mathbb{Z}_p^*$. Znając $(g^x \bmod p)$ trudno jest wyznaczyć $x$. Można pokazać, że jeśli ten problem jest trudny w najgorszym przypadku, to jest też trudny w średnim przypadku.
- I inne (np. bazujące na krzywych eliptycznych)

Chcemy dowodzić bezpieczeństwa protokołów kryptograficznych przy założeniu, że te problemy są trudne.

Tylko co to dokładnie znaczy, że one są trudne dla większości instancji? Potrzebujemy jakiejś definicji.

# Funkcje jednokierunkowe

Rodzina funkcji $f_n:\{0,1\}^n \rightarrow \{0,1\}^{m(n)}$ jest $(\varepsilon(n),s(n))$-jednokierunkowa, jeśli jest obliczalna w czasie wielomianowym, a dla dowolnego algorytmu randomizowanego $A$ działającego w czasie $s(n)$

$Pr[A$ odwraca $f_n(x)] \leq \varepsilon(n)$

- „odwraca” tzn. znając $f_n(x)$ znajduje $y$ taki, że $f_n(y)=f_n(x)$
- prawdopodobieństwo jest po ciągach wejściowych $x \in \{0,1\}^n$ oraz po losowaniach wykonanych przez $A$
- nie możemy zabronić przeciwnikowi korzystać z bitów losowych
- możnaby nawet założyć, że $A$ to algorytm niejednorodny, zależny od $n$ (czyli z klasy **P/poly**)

# Funkcje jednokierunkowe

Rodzina funkcji $f_n:\{0,1\}^n \rightarrow \{0,1\}^{m(n)}$ jest $(\varepsilon(n),s(n))$-jednokierunkowa, jeśli jest obliczalna w czasie wielomianowym, a dla dowolnego algorytmu randomizowanego $A$ działającego w czasie $s(n)$

$Pr[A$ odwraca $f_n(x)] \leq \varepsilon(n)$

- chcemy, aby $s(n)$ rosło szybciej od dowolnego wielomianu; przypuszczamy, że podanych wcześniej problemów nie można rozwiązać w czasie podwykładniczym, tzn. szybciej niż $2^{n^c}$ dla jakiejś (być może małej) stałej $c>0$
- prawdopodobieństwo $\varepsilon(n)$ powinno maleć wraz z $n$ (np. wielomianowo)
- Twierdzenie (Yao): jeśli istnieją funkcje jednokierunkowe dla $\varepsilon(n)=1-1/n^c$ dla pewnego $c$ (tzw. słabe funkcje jednokierunkowe, które są trudne tylko dla $1/n^c$ wejść), to istnieją funkcje jednokierunkowe dla $\varepsilon(n)<1/n^k$ dla każdego $k$ (tzw. silne fun. jednok.)

# Funkcje jednokierunkowe

Jak zapisać nasze problemy jako funkcje jednokierunkowe:

- Logarytm dyskretny. Niech $p_1, p_2, \ldots$ to ciąg liczb pierwszych, gdzie $p_i$ ma $i$ bitów. Niech $g_i$ to generator $\mathbb{Z}^*_{p_i}$. Wtedy $i$-ta funkcja to $x \to (g_i^x \bmod p_i)$.
- Faktoryzacja. Wiadomo, że istnieje wielomianowy (względem $n$) algorytm randomizowany generujący liczby pierwsze długości $n$. Możemy go rozważać jako deterministyczny algorytm przekształcający ciąg bitów losowych $r$ na liczbę pierwszą $A(r)$ długości $n$. Wtedy rozważana funkcja (przypuszczalnie jednokierunkowa) to: $(r_1, r_2) \to A(r_1) \cdot A(r_2)$
- Szyfrowanie RSA to też funkcja (przypuszczalnie jednokierunkowa), itp.

# Generatory pseudolosowe

Najlepszy szyfr (one-time pad):

- Nadawca i odbiorca znają wspólny (jednorazowy) klucz $x$ tej samej długości co wiadomość $m$
- Nadawca wysyła ($m$ XOR $x$)
- Odbiorca oblicza (($m$ XOR $x$) XOR $x$)=$m$
- Pełne bezpieczeństwo nie znając $x$, przeciwnik widząc $(m\ XOR\ x)$ nie wie nic o $m$ (niezależnie od mocy obliczeniowej)

# Generatory pseudolosowe

Najlepszy szyfr (one-time pad):

- Nadawca i odbiorca znają wspólny (jednorazowy) klucz $x$ tej samej długości co wiadomość $m$
- Nadawca wysyła ($m$ XOR $x$)
- Odbiorca oblicza (($m$ XOR $x$) XOR $x$)=$m$
- Pełne bezpieczeństwo nie znając $x$, przeciwnik widząc $(m\ XOR\ x)$ nie wie nic o $m$ (niezależnie od mocy obliczeniowej)
- Problem: strony musiałyby wcześniej ustalić wspólnie bardzo długi klucz
- Powiązany problem: skąd w ogóle wziąć tak długi losowy ciąg
- Rozwiązanie: generator pseudolosowy – na podstawie krótkiego (znanego obu stronom) klucza, wygeneruj długi ciąg, który wygląda jak losowy

# Generatory pseudolosowe

Definicja:

- Rozważamy rodzinę funkcji $g_n:\{0,1\}^n \rightarrow \{0,1\}^{m(n)}$ obliczalnych w czasie wielomianowym
- Jest to generator $(\varepsilon(n),s(n))$-pseudolosowy, jeśli dla dowolnego algorytmu randomizowanego $A$ działającego w czasie $s(n)$, dla każdego dużego $n$:

$$|Pr_{x\in\{0,1\}^n}[A(g_n(x))=1]-Pr_{y\in\{0,1\}^n}[A(y)=1]|\leq\varepsilon(n)$$

(czyli wygenerowanego ciągu pseudolosowego nie da się odróżnić od ciągu losowego)

# Generatory pseudolosowe

Definicja:

- Rozważamy rodzinę funkcji $g_n:\{0,1\}^n \rightarrow \{0,1\}^{m(n)}$ obliczalnych w czasie wielomianowym
- Jest to generator $(\varepsilon(n),s(n))$-pseudolosowy, jeśli dla dowolnego algorytmu randomizowanego $A$ działającego w czasie $s(n)$, dla każdego dużego $n$:

$$|Pr_{x\in\{0,1\}^n}[A(g_n(x))=1]-Pr_{y\in\{0,1\}^n}[A(y)=1]|\leq\varepsilon(n)$$

(czyli wygenerowanego ciągu pseudolosowego nie da się odróżnić od ciągu losowego)

- Równoważna definicja:

$$Pr_{x\in\{0,1\}^n}[A(g_n(x)[1..i])=g_n(x)[i+1]]\leq\varepsilon(n)$$

(czyli widząc $i$ bitów ciągu pseudolosowego nie da się wyznaczyć kolejnego bitu)

Równoważność nie jest oczywista, ale też nie jest trudna

# Generatory pseudolosowe

Twierdzenie: Istnieją funkcje jednokierunkowe ⇔ istnieją generatory pseudolosowe

Implikacja ⇐ w zasadzie oczywista: generator pseudolosowy musi być funkcją jednokierunkową

Implikacja ⇒ dużo trudniejsza (odpowiednio wykorzystując funkcję jednokierunkową można stworzyć generator pseudolosowy)

# Dowody z wiedzą zerową

Dowód z wiedzą zerową – procedura, w której jedna ze stron potrafi udowodnić drugiej, że dysponuje pewną informacją, bez jej ujawniania

# Dowody z wiedzą zerową

Dowód z wiedzą zerową – procedura, w której jedna ze stron potrafi udowodnić drugiej, że dysponuje pewną informacją, bez jej ujawniania:

- Jeśli dowodzący dysponuje daną informacją, może zawsze przekonać o tym weryfikującego
- Jeśli dowodzący nie dysponuje daną informacją, może oszukać weryfikującego, że nią dysponuje, jedynie z małym prawdopodobieństwem
- Weryfikujący (nawet oszukując tzn. nie stosując się do protokołu) nie dowie się nic o danej informacji (to można odpowiednio sformalizować)
- Obie strony mają ograniczoną moc obliczeniową (to zwykłe algorytmy, działające w czasie wielomianowym, mając dostęp do bitów losowych).

# Dowody z wiedzą zerową

Przykład – izomorfizm grafów:

- P twierdzi, że zna izomorfizm między $G_1$ i $G_2$
- V chce się o tym przekonać
- P wysyła graf $H$
- V wysyła liczbę $i \in \{1,2\}$
- P wysyła izomorfizm między $H$ a $G_i$

# Dowody z wiedzą zerową

Przykład – izomorfizm grafów:

- P twierdzi, że zna izomorfizm między $G_1$ i $G_2$
- V chce się o tym przekonać
- P wysyła graf $H$
- V wysyła liczbę $i \in \{1,2\}$
- P wysyła izomorfizm między $H$ a $G_i$
- Jeśli P zna izomorfizm między $G_1$ i $G_2$, generuje $H$ przez dowolną zamianę etykietek wierzchołków któregoś z grafów. Następnie z łatwością może wygenerować izomorfizm do jednego lub drugiego grafu.
- Jeśli P nie zna izomorfizmu między $G_1$ i $G_2$, to nie potrafi znaleźć takiego $H$, żeby mógł zbudować izomorfizm zarówno do $G_1$ jak i do – gdyby znał taki $H$ mógłby zbudować izomorfizm między $G_1$ i $G_2$.
- Znajomość izomorfizmu między $H$ i $G_i$ nie ułatwia znalezienia izomorfizmu między $G_1$ i $G_2$

# Złożoność Kołmogorowa

Idea: niektóre liczby (napisy itp.) łatwiej zapamiętać niż inne.
Czyli są bardziej złożone.
To zależy nie tylko od wielkości liczby.

$\underbrace{100\ldots0}_{100}$

$100^{\overbrace{100^{\cdots^{100}}}^{100}}$

25839496603316858921

31415926535897932384

# Złożoność Kołmogorowa

Idea: niektóre liczby (napisy itp.) łatwiej zapamiętać niż inne.
Czyli są bardziej złożone.
To zależy nie tylko od wielkości liczby.

$\underbrace{100\ldots0}_{100}$

$100^{100^{\cdot^{\cdot^{\cdot^{100}}}}}$ (100 poziomów)

25839496603316858921 ← 20 „losowych” cyfr

31415926535897932384 ← $\pi$

$$\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots=\frac{\pi}{4}$$

# Złożoność Kołmogorowa

Idea: złożoność liczby = długość jej najkrótszego opisu

Definicja: złożoność liczby = liczba stanów najmniejszej maszyny Turinga wypisującej tą liczbę (przy pustym wejściu)

# Złożoność Kołmogorowa

Idea: złożoność liczby = długość jej najkrótszego opisu

Definicja: złożoność liczby = liczba stanów najmniejszej maszyny Turinga wypisującej tą liczbę (przy pustym wejściu)

Paradoks Berry'ego: niech $n$ będzie najmniejszą liczbą, której nie da się zdefiniować za pomocą ≤100 słów (właśnie ją zdefiniowaliśmy)

# Złożoność Kołmogorowa

Idea: złożoność liczby = długość jej najkrótszego opisu

Definicja: złożoność liczby = liczba stanów najmniejszej maszyny Turinga wypisującej tą liczbę (przy pustym wejściu)

Paradoks Berry'ego: niech $n$ będzie najmniejszą liczbą, której nie da się zdefiniować za pomocą ≤100 słów (właśnie ją zdefiniowaliśmy)

Twierdzenie

Funkcja przypisująca liczbie jej złożoność nie jest obliczalna.

Dowód

Jeśli byłaby obliczalna, to moglibyśmy także obliczać funkcję:

$k \rightarrow$ najmniejsza liczba $n_k$ mająca złożoność $\geq k$

(liczymy złożoność kolejnych liczb, aż dojdziemy do takiej, której złożoność jest $\geq k$)

Ale to oznacza, że złożoność $n_k$ jest $\leq C+log(k)$, dla pewnej stałej $C$:

wypisujemy liczbę $k$, po czym aplikujemy funkcję $k \rightarrow n_k$

Czyli dla każdego $k$ mamy $k \leq C+log(k)$ – sprzeczność

# Złożoność komunikacyjna

Złożoność komunikacyjna:

- Dana jest funkcja $f:X\times Y\rightarrow Z$ (zazwyczaj $X=Y=\{0,1\}^n$ $Z=\{0,1\}$).
- Alicja zna $x\in X$, Bob zna $y\in Y$.
- Ile bitów muszą sobie przekazać, aby Alicja poznała $f(x,y)$?

Oczywiście $n$ bitów zawsze wystarczy, ale może dla niektórych funkcji da się mniej.

# Złożoność komunikacyjna

Złożoność komunikacyjna:

- Dana jest funkcja $f:X\times Y\to Z$ (zazwyczaj $X=Y=\{0,1\}^n$ $Z=\{0,1\}$).
- Alicja zna $x\in X$, Bob zna $y\in Y$.
- Ile bitów muszą sobie przekazać, aby Alicja poznała $f(x,y)$?

Oczywiście $n$ bitów zawsze wystarczy, ale może dla niektórych funkcji da się mniej.

Przykład: funkcja „czy $x=y$?” ma wymaga przesłania $n$ bitów.
Dowód: Załóżmy, że wystarczy wysłać $n-1$ bitów. Wówczas istnieją dwie pary $(x,x)$ i $(x',x')$ dla których komunikacja wygląda identycznie. Ale dla pary $(x,x')$ też przebiega tak samo, czyli wynik będzie źle obliczony.

# Złożoność komunikacyjna

Złożoność komunikacyjna:

- Dana jest funkcja $f:X\times Y\to Z$ (zazwyczaj $X=Y=\{0,1\}^n$ $Z=\{0,1\}$).
- Alicja zna $x\in X$, Bob zna $y\in Y$.
- Ile bitów muszą sobie przekazać, aby Alicja poznała $f(x,y)$?

Oczywiście $n$ bitów zawsze wystarczy, ale może dla niektórych funkcji da się mniej.

Przykład: funkcja „czy $x=y$?” ma wymaga przesłania $n$ bitów.
Dowód: Załóżmy, że wystarczy wysłać $n-1$ bitów. Wówczas istnieją dwie pary $(x,x)$ i $(x',x')$ dla których komunikacja wygląda identycznie. Ale dla pary $(x,x')$ też przebiega tak samo, czyli wynik będzie źle obliczony.

Dolne ograniczenia na złożoność komunikacyjną różnych funkcji zostały użyte do udowodnienia dolnych ograniczeń na złożoność pewnych problemów, np. dla algorytmów strumieniowych.
Patrz także: zadanie 1.5.3 – maszyna jednotaśmowa rozpoznająca język palindromów potrzebuje czasu $\Omega(n^2)$

# Obliczenia kwantowe

- nowy model obliczeniowy
- zakłada się, że w przyszłości będzie można je fizycznie wykonywać (w przeciwieństwie do np. obliczeń niedeterministycznych lub alternujących) – chociaż obecnie w zasadzie nie istnieją, to wydaje się, że prawa fizyki dopuszczają ich istnienie
- chociaż komputery kwantowe nie istnieją, matematycznie model obliczeń kwantowych jest dobrze zdefiniowany

# Obliczenia kwantowe

- nowy model obliczeniowy
- zakłada się, że w przyszłości będzie można je fizycznie wykonywać (w przeciwieństwie do np. obliczeń niedeterministycznych lub alternujących) – chociaż obecnie w zasadzie nie istnieją, to wydaje się, że prawa fizyki dopuszczają ich istnienie
- chociaż komputery kwantowe nie istnieją, matematycznie model obliczeń kwantowych jest dobrze zdefiniowany
- być może w pewnych obliczeniach komputery kwantowe mają wykładniczą przewagę nad obliczeniami klasycznymi (deterministycznymi lub randomizowanymi)
- zatem potencjalnie przeczą silnej tezie Churcha-Turinga (każde fizycznie realizowalne urządzenie obliczeniowe można zasymulować na maszynie Turinga z narzutem wielomianowym)

# Obliczenia kwantowe

Jak liczymy?

- $m$ kubitów to wektor $v$ długości $1$ z $\mathbb{C}^{2^m}$
- operacja to przemnożenie wektora $v$ przez ustaloną macierz unitarną (tzn. nie zmieniającą długości wektora), wybraną spośród dostępnych macierzy
- odczyt: wartość $i \in \{0,1\}^m$ jest odczytana z prawdopodobieństwem $|v_i|^2$ (nie możemy odczytać całego wektora $v$)

# Obliczenia kwantowe

Jak liczymy?

- $m$ kubitów to wektor $v$ długości $1$ z $\mathbb{C}^{2^m}$
- operacja to przemnożenie wektora $v$ przez ustaloną macierz unitarną (tzn. nie zmieniającą długości wektora), wybraną spośród dostępnych macierzy
- odczyt: wartość $i \in \{0,1\}^m$ jest odczytana z prawdopodobieństwem $|v_i|^2$ (nie możemy odczytać całego wektora $v$)

Jak to się ma do obliczenia klasycznego?

- klasyczne obliczenie, w którym zawartość pamięci to $i \in \{0,1\}^m$ odpowiada sytuacji kiedy $v_i = 1$ oraz $v_j = 0$ $\forall j \neq i$
- zatem (intuicyjnie): w obliczeniu kwantowym przeprowadzamy to samo obliczenie dla wielu możliwych zawartości pamięci równocześnie, na koniec losowany jest jeden z otrzymanych wyników

# Obliczenia kwantowe

Co można obliczyć?

- Niech **BQP** – języki rozpoznawalne na komputerze kwantowym w czasie wielomianowym (definiowane podobnie jak **BPP** – prawdopodobieństwo uzyskania poprawnego wyniku $>3/4$)
- Łatwo dowieść, że **BPP**$\subseteq$**BQP**
- Z drugiej strony **BQP**$\subseteq$**PSPACE**

# Obliczenia kwantowe

Co można obliczyć?

- Niech **BQP** – języki rozpoznawalne na komputerze kwantowym w czasie wielomianowym (definiowane podobnie jak **BPP** – prawdopodobieństwo uzyskania poprawnego wyniku $>3/4$)
- Łatwo dowieść, że **BPP**$\subseteq$**BQP**
- Z drugiej strony **BQP**$\subseteq$**PSPACE**
- W **BQP** umiemy rozkładać liczby na czynniki pierwsze (algorytm Shor'a z 1994) – mimo że klasycznie nie umiemy!
- Ogólne sformułowanie problemów poszukiwania (**NP**): dana jest funkcja $f:\{0,1\}^n \rightarrow \{0,1\}$ obliczalna w czasie wielomianowym, czy istnieje $a$ takie, że $f(a)=1$? Klasycznie potrzeba $2^n$ wywołań $f$, kwantowo wystarczy $2^{n/2}$ (kwadratowo mniej) – algorytm Grover'a

# Obliczenia kwantowe

Co można obliczyć?

- Niech **BQP** – języki rozpoznawalne na komputerze kwantowym w czasie wielomianowym (definiowane podobnie jak **BPP** – prawdopodobieństwo uzyskania poprawnego wyniku $>3/4$)
- Łatwo dowieść, że **BPP**$\subseteq$**BQP**
- Z drugiej strony **BQP**$\subseteq$**PSPACE**
- W **BQP** umiemy rozkładać liczby na czynniki pierwsze (algorytm Shor'a z 1994) – mimo że klasycznie nie umiemy!
- Ogólne sformułowanie problemów poszukiwania (**NP**): dana jest funkcja $f:\{0,1\}^n \rightarrow \{0,1\}$ obliczalna w czasie wielomianowym, czy istnieje $a$ takie, że $f(a)=1$? Klasycznie potrzeba $2^n$ wywołań $f$, kwantowo wystarczy $2^{n/2}$ (kwadratowo mniej) – algorytm Grover'a
- Jeśli $f$ jest dana jako wyrocznia, to nie da się mniej (tzn. **NP**$^A \not\subseteq$**BQP**$^A$ dla pewnej wyroczni $A$)
- Zatem komputer kwantowy może szybko rozwiązać problem poszukiwania jedynie wykorzystując strukturę tego problemu (jak dla szukania czynników pierwszych).